09/856515
0500

Copy for the Elected Office (EO/US)

PATENT COOPERATION TREATY

PCT

NOTIFICATION OF THE RECORDING OF A CHANGE

(PCT Rule 92bis.1 and
Administrative Instructions, Section 422)

From the INTERNATIONAL BUREAU

To:

GREENE-KELLY, James, Patrick
Lloyd Wise
Tanjong Pagar
P.O. Box 636
Singapore 910816
SINGAPOUR

Date of mailing (day/month/year) 25 July 2001 (25.07.01)	
Applicant's or agent's file reference FP1123	**IMPORTANT NOTIFICATION**
International application No. PCT/SG99/00018	**International filing date** (day/month/year) 18 March 1999 (18.03.99)

1. The following indications appeared on record concerning:

☒ the applicant ☒ the inventor □ the agent □ the common representative

Name and Address PANG, Hwee, Hwa 19 Shelford Road #01-42 Singapore 288408 Singapore	State of Nationality SG	State of Residence SG
	Telephone No.	
	Facsimile No.	
	Teleprinter No.	

2. The International Bureau hereby notifies the applicant that the following change has been recorded concerning:

□ the person □ the name ☒ the address □ the nationality □ the residence

Name and Address PANG, Hwee, Hwa 201 Tanjong Rhu Road #15-11 Singapore 436917 Singapore	State of Nationality SG	State of Residence SG
	Telephone No.	
	Facsimile No.	
	Teleprinter No.	

3. Further observations, if necessary:

4. A copy of this notification has been sent to:

☒ the receiving Office
□ the International Searching Authority
☒ the International Preliminary Examining Authority
□ the designated Offices concerned
☒ the elected Offices concerned
□ other:

The International Bureau of WIPO **34, chemin des Colombettes** **1211 Geneva 20, Switzerland** Facsimile No.: (41-22) 740.14.35	Authorized officer Dominique DELMAS Telephone No.: (41-22) 338.83.38

P⸍ ⸌ENT COOPERATION TREAT⸌

From the INTERNATIONAL BUREAU

PCT

NOTIFICATION OF ELECTION

(PCT Rule 61.2)

To:

Assistant Commissioner for Patents
United States Patent and Trademark
Office
Box PCT
Washington, D.C.20231
ETATS-UNIS D'AMERIQUE

in its capacity as elected Office

Date of mailing (day/month/year) 22 September 2000 (22.09.00)	
International application No. PCT/SG99/00018	**Applicant's or agent's file reference** FP1123
International filing date (day/month/year) 18 March 1999 (18.03.99)	**Priority date** (day/month/year) 16 December 1998 (16.12.98)
Applicant NGAIR, Teow, Hin et al	

1. The designated Office is hereby notified of its election made:

☒ in the demand filed with the International Preliminary Examining Authority on:

30 June 2000 (30.06.00)

□ in a notice effecting later election filed with the International Bureau on:

2. The election ☒ was

□ was not

made before the expiration of 19 months from the priority date or, where Rule 32 applies, within the time limit under Rule 32.2(b).

The International Bureau of WIPO **34, chemin des Colombettes** **1211 Geneva 20, Switzerland** Facsimile No.: (41-22) 740.14.35	Authorized officer Lazar Joseph Panakal Telephone No.: (41-22) 338.83.38

PATENT COOPERATION TREATY

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

Applicant's or agent's file reference FP1123	**FOR FURTHER ACTION**	See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)
International application No. PCT/SG 99/00018	International filing date (*day/month/year*) 18 March 1999 (18.03.1999)	Priority Date (*day/month/year*) 16 December 1998 (16.12.1998)
International Patent Classification (IPC) or national classification and IPC IPC^{7}: 06F 9/46, G06F 9/54		
Applicant Kent Ridge Digital Labs et al.		

1. This international preliminary examination report has been prepared by this International Preliminary Examination Authority and is transmitted to the applicant according to Article 36.

2. This REPORT consists of a total of 6 sheets, including this cover sheet.

   □ This report is also accompanied by ANNEXES, i.e., sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

   These annexes consist of a total of ____________ sheets.

3. This report contains indications relating to the following items:

   I. ☒ Basis of the opinion

   II. □ Priority

   III. □ Non-establishment of opinion with regard to novelty, inventive step and industrial applicability

   IV. □ Lack of unity of invention

   V. ☒ Reasoned statement under Rule 66.2(a)(ii) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

   VI. □ Certain documents cited

   VII. □ Certain defects in the international application

   VIII. □ Certain observations on the international application

Date of submission of the demand 30.06.2000	Date of completion of this report 24 July 2003 (24.07.2003)
Name and mailing address of the IPEA/AT Austrian Patent Office Dresdner Straße 87 A-1200 Vienna Facsimile No. 1/53424/200	Authorized officer FASTENBAUER K. Telephone No. 1/53424/447

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.
PCT/SG 99/00018

I. Basis of the report

1. With regard to the **elements** of the international application:*

☒ the international application as originally filed

□ the description:
pages ______, as originally filed
pages ______, filed with the demand
pages ______, filed with the letter of ______.

□ the claims:
pages ______, as originally filed
pages ______, as amended (together with any statement) under Article 19
pages ______, filed with the demand
pages ______, filed with the letter of ______.

□ the drawings:
pages ______, as originally filed
pages ______, filed with the demand
pages ______, filed with the letter of ______.

□ the sequence listing part of the description:
pages ______, as originally filed
pages ______, filed with the demand
pages ______, filed with the letter of ______.

2. With regard to the **language**, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.
These elements were available or furnished to this Authority in the following language ______ which is:

□ the language of a translation furnished for the purposes of international search (under Rule 23.1(b)).

□ the language of publication of the international application (under Rule 48.3(b)).

□ the language of the translation furnished for the purposes of international preliminary examination (under Rule 55.2 and/ or 55.3).

3. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

□ contained in the international application in printed form.

□ filed together with the international application in computer readable form.

□ furnished subsequently to this Authority in written form.

□ furnished subsequently to this Authority in computer readable form.

□ The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

□ The statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished.

4. □ The amendments have resulted in the cancellation of:

□ the description, pages ______.

□ the claims, Nos. ______.

□ the drawings, sheets/fig ______.

5. □ This report has been established as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed, as indicated in the Supplemental Box (Rule 70.2(c)).**

* *Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as „originally filed" and are not annexed to this report since they do not contain amendments (Rules 70.16 and 70.17).*

** *Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.*

IV. Lack of unity of invention

1. In response to the invitation to restrict or pay additional fees the applicant has:

□ restricted the claims.

□ paid additional fees.

□ paid additional fees under protest.

□ neither restricted nor paid additional fees.

2. □ This Authority found that the requirements of unity of invention is not complied with and chose, according to Rule 68.1, not to invite the applicant to restrict or pay additional fees.

3. This Authority considers that the requirements of unity of invention in accordance with Rules 13.1, 13.2 and 13.3 is

□ complied with.

☒ not complied with for the following reasons:

cl 1. a method for migrating a computing process discards data
(+31,32) (discard /add library modules and/or device driver)
cl. 2. . . . discards data . . . prior to migration
cl 3. . . . a contruct is formded . . .
cl 4. . . . suspend all active threads . . .
cl. 5. . . . falling within lists
---
cl 6 . . . authorizing signature
cl 7. . . . sent directly to said second host
cl 8. . . . sent to an intermediate storage means . . .
cl 9. . . . second process is run on said second host . . .
cl. 10 . . . third process . . .

the features of cl. 6-10 do not refer to an inventive step common to the features of cl. 4-5. They could only be grouped to a unique invention when the features of claim 1-3 to which they refer, could be considered as being new or involving an inventive step. As claim 1-3 are neighter new nor inventive, claim 6-10 present an a-posteriori lack of unity of invention.

4. Consequently, the following parts of the international application were the subject of international preliminary examination in establishing this opinion:

☒ all parts.

□ the parts relating to claims Nos. _______ .

V. Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Claims	11-30	YES
	Claims	1-10,31,32	NO
Inventive step (IS)	Claims	11-30	YES
	Claims	1-10,31,32	NO
Industrial applicability (IA)	Claims	1-32	YES
	Claims	----	NO

Citations and explanations (Rule 70.7)

The following X-documents are cited in the search report 10-9-01 ds

D1) US 5,603,031 A (HELGERSON et al.),11,2,1997

D2) ARIDOR, Yariv and LANGE Danny, Agent design pattern: elements of agent application design. 10.9.01 ds

D2 discloses some reusable agent design patterns, especially the Locker-pattern. See page 110, col 1: ... *Agents can exploit the LOCKER pattern to temporarily store data in private. In this way, they can avoid bringing along data that for the moment are not needed. On a later occasion, agents can return and retrieve the private data stored in the locker. For example, in an agent-based purchasing system, an agent may visit a dealer's host outside the company network. In this case, it can* ***store its sensitive data in a locker before leaving*** *the company network. The result is a reduction of network traffic and improved data integrity...*

Hence, with the locker pattern, the agent discards data specific to the "company network" before leaving this network and moving to the dealer's host. It is inherent to the application (purchasing) that the agent will receive data "specific to the dealer's host" as this is the reason why it is sent to the dealer's host. Therefore D2 discloses all features of claim 1, and also of claim 2, as it mentions that the specific data are stored "before leaving the network company".

D1 discloses, as the applicant mentions in his letter dated 10 July 2001, a concept where an agent moving from one place to another discards objects which already exist in the second place before leaving the first place. As the applicant argues ... *parts of the agent which are not specific to the first place ... are discarded before the agent moves, and parts of the agent which are not specific to the second place ... are added after it has moved ...*

Hence the applicant agrees that it belongs to the state of the art to discard objects before and adding objects after the agent moves.

Supplemental Box
(To be used when the space in any of the preceding boxes is not sufficient)

Continuation of: Box V (page 1)

He argues that the present invention requires that the discarded / added objects are *specific* to the first / second host, which following his arguments means that those objects would not function in the second / first host. In other words, the applicant argues that the *inventive step* in view of D1 consists in *not transporting useless objects from a first place to a second place.*

In everyday life, the common sense leads every people to not transport useless objects from a first place to a second place. No person making any kind of journey would take with her objects that she knows to be useless on the destination. So that what the applicant says to be the fundamental idea of the present application and the difference to the state of the art, would be interpreted as an obvious behaviour of common-sensed people in everyday life.

Therefore,

- the applicant agrees that D1 discloses a way of discarding objects before and adding objects after the agent moves
- this concerns objects that the agent (perhaps) could use on the start / destination host
- the common sense says that one does not transport objects that one could not use on the destination

So, a person skilled in the art knowing D1 and using her common sense would obviously have the idea of discarding useless objects before the agent moves. Knowing the locker pattern of D2, she also would come to the same result of her considerations.

The other documents cited in the search report as A-documents disclose that "discarding" objects that are not used at the moment is not only common-sense behaviour in everyday life, but also well-known in the art of computer programming. E.g. "unloading and unlinking objects" on the host-system BS2000/OSD discloses a way of discarding useless objects (program modules). Zipping data / programs also is a way of discarding useless parts of the data / program before moving them from a first host to a second host, and is known by nearly every computer-user, even without any particular programming-knowledge.

In the multiagent application development system JAFMAS, (p. 104, chapter 5.2.1 "Gathering Required Resources) ... *each agent is autonomous in nature when it becomes on line...finding required resources for agents is done by the RequestedResrcProvider and ReqdResource classes. The RequestedResrcProviderkeeps a* **list** *of all the resources requested by a parent agent and for each resource it creates a ReqdResource object ...*

This shows that such considerations of common sense in everyday life are in the same way valuable in the art of computer programming, and that there is no inventive step needed to adopt those ideas for e.g. mobile agents.

The additional features claimed in claims 2-10 are known or induced by D1.

Hence **claim 1-10** cannot be seen as being new or involving an inventive step in view of D1, D2 and the common sense.

The feature of discarding objects specific to the first host *after* the movement are at least partly against the common sense and can therefore be seen as inventive in view of the cited documents. Therefore the **claims 11-20** (discard after the movement, add after discarding) **and 21-30** (discard

Supplemental Box
(To be used when the space in any of the preceding boxes is not sufficient)

Continuation of: Box V (page 2)

after the movement / add before discarding) can be seen as new and involving an inventive step in view of the cited documents. It nevertheless has to be mentioned that the considerations about the timing of discarding / adding objects merely are the result of a simple evaluation of all possible cases: discard before / after moving; add after moving and before/after discarding. Such systematically evaluations of all possible cases mostly do not present highly-inventive steps.

The features of **claim 31 and 32** cannot be considered as being new of involving an inventive step in view of the cited documents and the common sense as long as they refer to claims 1-10. It is well-known behaviour of any kind of program to take e.g. device drivers from the host where the program is running, hence the "second" host. For example if a first person sends an e-mail to a second person, this second person reads the e-mail on a second computer and prints the e-mail, the printer-driver are obviously taken from this second computer, and never from the first computer. It is the same in the case where a program is moved from one host to another: to run, it will obviously use the device drivers from the second computer.

Especially as "device drivers" are the standard case of objects used from the destination computer, as mostly any other drivers brought along from the source computer would not work on the destination.

It is also known since the so called "dinosaur"-host-systems like BS2000/OSD to have e.g. own developer-computer with the complete compiler + test environment. Programs developed on this system are later on moved to the production system, which only comprises the so-called "run-time-libraries". Those "run-time-libraries" are not moved from the development to the production system. The principle of such a development / production environment is basically the same as the idea of leaving the "library modules" on the first host and using the "library modules" on the second host.

There is no reason why an agent should try another behaviour as this well-known best practice in the art of computer programming.

The industrial applicability is given for all claims.

INTERNATIONAL SEARCH REPORT

International application No.
PCT / SG 99/00018

A. CLASSIFICATION OF SUBJECT MATTER

IPC^6: G06F 9/46, G06F 9/54

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC^6: G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPODOC, WPI, The INTERNET, ACM Digital Library

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 5603031A (HELGESON et al.) 11 February 1997 (11.2.1997), summary of the invention, col. 7 li. 66 - col.8 li. 10, col. 8 li. 51 ff., col. 10 li. 31- col. 11 li. 28 , fig. 24A-C, totality.	1-10, 31,32 11-30
X A	ARIDOR, Yariv, and LANGE Danny B. Agent design patterns: elements of agent application design. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p.108-115 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/articles/proceedings/ai /280765/p108-aridor/p108-aridor.pdf>.	1-3,7,9,31,32 4-6,8,10-30
A	CHEN Qiming, CHUNDI Parvathi, DAYAL Umeshwar, and HSU Meichun. Dynamic software agents for business intelligence applications (poster). Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p.453-454 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:<http://www.acm.org/pubs/ articles/ proceedings/ai/280765/p453-chen/p453-chen.pdf>.	1-32

☒ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:

„A“ document defining the general state of the art which is not considered to be of particular relevance

„E“ earlier application or patent but published on or after the international filing date

„L“ document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

„O“ document referring to an oral disclosure, use, exhibition or other means

„P“ document published prior to the international filing date but later than the priority date claimed

„T“ later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

„X“ document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

„Y“ document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

„&“ document member of the same patent family

Date of the actual completion of the international search 18 May 2000 (18.05.2000)	Date of mailing of the international search report 19 May 2000 (19.05.00)
Name and mailing adress of the ISA/AT Austrian Patent Office Kohlmarkt 8-10; A-1014 Vienna Facsimile No. 1/53424/535	Authorized officer Fastenbauer Telephone No. 1/53424/447

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CHAUHAN Deepika and BAKER Albert D. JAFMAS: a multiagent application development system. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p. 100-107 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/citations/proceedings /ai/280765/p100-chauhan/ p100-chauhan.pdf>.	1-32
A	PKZIP (R) FAST! Create/Update Utility Version 2.04g, 1993-01-02. Copyright 1989-1993 PKWARE Inc. Shareware Version.	1-32
A	WO 97/35262 A (HITACHI), 25 September 1997 (25.9.1997), totality.	1-32
A	BS2000/OSD-BC V1.0, Dynamic Binder Loader/Starter, Chapter 2.3.1: Unloading and unlinking objects (p.44), ID: U5137-J-Z125-2-7600 [online] April 1993 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL:http://manuals.mchp.siemens.de/servers/bs2_man/man_us/bs2_bc/ V1_0/bls.pdf>	1-10,31,32
A	openUTM V4.0 (BS2000/OSD), Concepts and Functions, Chapter 5: Structure of UTM Applications, especially 5.2: The process concept, para. 3, page 67 , ID: U20683-J-Z135-2-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http:// manuals. mchp. siemens.de/servers/bs2_man/man_us/utm/v4_0/ utm_kon.pdf>	1-10,31,32
A	BS2000/OSD-BC V1.0, Performance Handbook, Chapter 5.3.1: Managing the resource main memory, especially: Deactivation (p.248), Waiting Time runout Control (p.255),Paging management algorithms (pp.256-257), ID: U1794-J-Z125-6-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp.siemens.de/servers/bs2_man/man_us /bs2_bc/V1_0/perform.pdf>	1-10,31,32

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG 99/00018

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. □ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. □ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. □ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

cl. 1	...	a method for migrating a computing process ... discards data
cl. 2	...	... discards data ... prior to migration
cl. 3-4	...	... a construct is formed ... / suspend all active threads
cl. 5	...	... falling whithin lists
cl. 6	...	... authorizing signature
cl. 7	...	... sent directly to said second host
cl. 8	...	... sent to an intermediate storage means ...
cl. 9	...	... second process is run on said second host ...
cl. 10	...	... third process ...

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. □ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. □ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. □ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims: it is covered by claims Nos.:

Remark on Protest

□ The additional search fees were accompanied by the applicant's protest.

□ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG 99/00018

continuation of first sheet (1):

cl. 11-17	... discards data ... after migration
cl. 21-30	... discards data ... after migration ... and after receiving data ...
cl. 31	... relates to library modules and/or input/output drivers of said first host
cl. 32	... relates to library modules and/or input/output drivers of said second host

INTERNATIONAL SEARCH REPORT

Information on patent family members

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US A 5603031	11-02-1997	AU A1 72164/94	06-02-1995
		EP A2 634719	18-01-1995
		EP A3 634719	03-01-1996
		JP A2 7182174	21-07-1995
		JP T2 7509799	26-10-1995
		WO A1 9502219	19-01-1995
		US A 6016393	18-01-2000
WO A 9735262a		none	

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : **G06F 9/46, 9/54**	**A1**	**(11) International Publication Number:** **WO 00/36507** **(43) International Publication Date:** 22 June 2000 (22.06.00)

(21) International Application Number: PCT/SG99/00018

(22) International Filing Date: 18 March 1999 (18.03.99)

(30) Priority Data:
PCT/SG98/00102 16 December 1998 (16.12.98) SG

(71) Applicant *(for all designated States except US)*: KENT RIDGE DIGITAL LABS [SG/SG]; 21 Heng Mui Keng Terrace, Singapore 119613 (SG).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: NGAIR, Teow, Hin [SG/SG]; 334 Kang Ching Road #13-254, Singapore 610334 (SG). PANG, Hwee, Hwa [SG/SG]; 19 Shelford Road #01-42, Singapore 288408 (SG).

(74) Agent: GREENE-KELLY, James, Patrick; Lloyd Wise, Tanjong Pagar, P.O. Box 636, Singapore 910816 (SG).

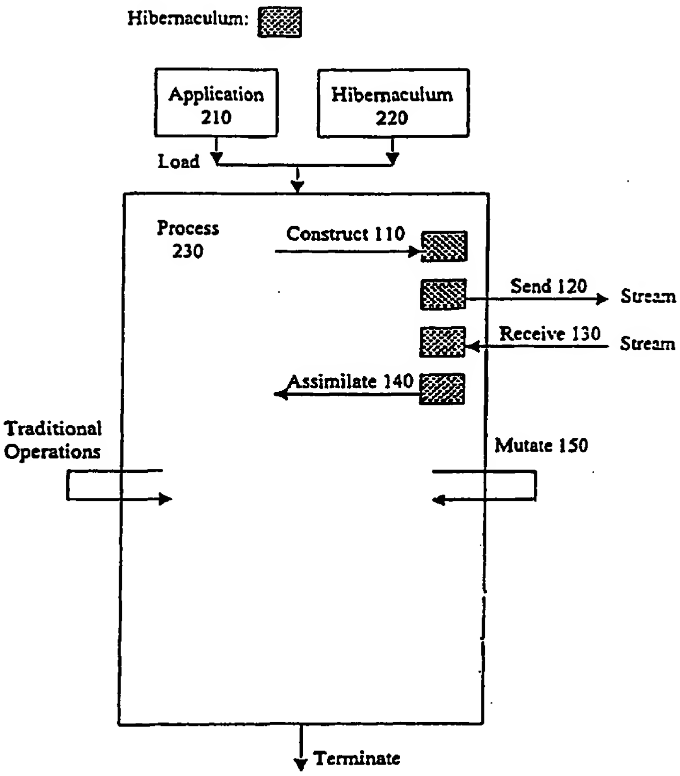
(81) Designated States: JP, SG, US.

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD FOR ADAPTING MIGRATING PROCESSES TO HOST MACHINES

(57) Abstract

A method is described for migrating a computing process from a first host to a second host, wherein prior to migration the process discards data, program code and execution states specific to the first host, and wherein after migration the process receives data, program code and execution states specific to the second host. This is achieved by forming a construct containing a process that is an application specific subset of the computing process to be transferred and then assimilating into that sub-process in the new host a system specific process relating to the new host. Alternatively the process may migrate intact, i.e. including the first host specific information, and then discard that information after migration either before or after assimilating information specific to the second host.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 :
G06F 9/46, 9/54

A1

(11) International Publication Number: **WO 00/36507**

(43) International Publication Date: 22 June 2000 (22.06.00)

(21) International Application Number: PCT/SG99/00018

(22) International Filing Date: 18 March 1999 (18.03.99)

(30) Priority Data:
PCT/SG98/00102 16 December 1998 (16.12.98) SG

(71) Applicant *(for all designated States except US)*: KENT RIDGE DIGITAL LABS [SG/SG]; 21 Heng Mui Keng Terrace, Singapore 119613 (SG).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: NGAIR, Teow, Hin [SG/SG]; 334 Kang Ching Road #13–254, Singapore 610334 (SG). PANG, Hwee, Hwa [SG/SG]; 19 Shelford Road #01–42, Singapore 288408 (SG).

(74) Agent: GREENE–KELLY, James, Patrick; Lloyd Wise, Tanjong Pagar, P.O. Box 636, Singapore 910816 (SG).

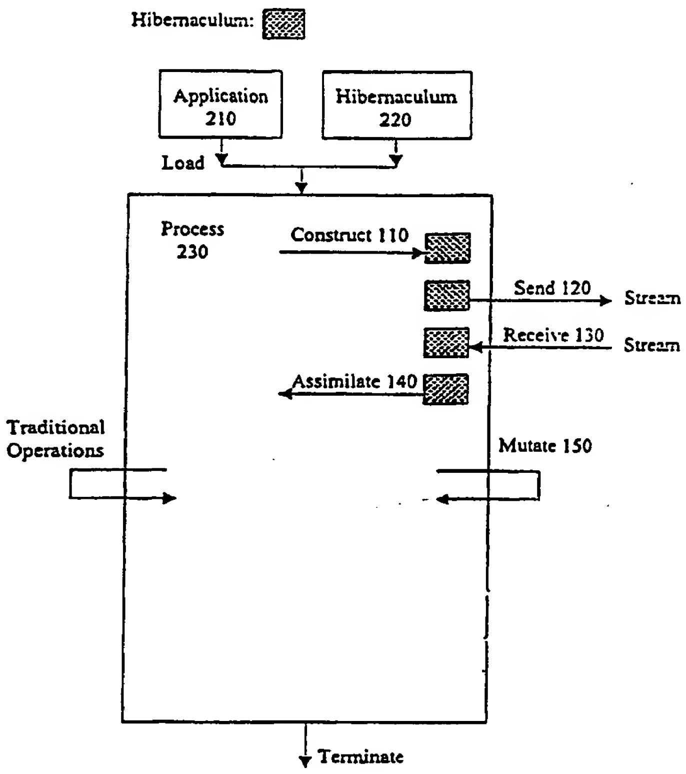
(81) Designated States: JP, SG, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD FOR ADAPTING MIGRATING PROCESSES TO HOST MACHINES

(57) Abstract

A method is described for migrating a computing process from a first host to a second host, wherein prior to migration the process discards data, program code and execution states specific to the first host, and wherein after migration the process receives data, program code and execution states specific to the second host. This is achieved by forming a construct containing a process that is an application specific subset of the computing process to be transferred and then assimilating into that sub–process in the new host a system specific process relating to the new host. Alternatively the process may migrate intact, i.e. including the first host specific information, and then discard that information after migration either before or after assimilating information specific to the second host.



*(Referred to in PCT Gazette No. 35/2000, Section II)

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

METHOD FOR ADAPTING MIGRATING PROCESSES TO HOST MACHINES

This invention relates to a method for adapting a migrating process as it moves from one machine to another machine with potentially differing hardware configurations.

Recent years have seen a number of developments in computing science regarding how elements within a software application are treated and handled. In this context the most basic elements to be found within a software application are data and program modules. Traditional procedural programming paradigms focus on the logic of the software application, so a program is structured based on program modules. One uses the program modules by explicitly supplying to them the data on which they should operate.

More recently there has been a move towards an object-oriented paradigm. In this paradigm, programs are structured around objects, with each object representing an entity in the world being modeled by the software application. Each object manages its own data (state), which are hidden from the external world (other objects and programs). An object in a program interacts with another object by sending it a message to invoke one of its exposed program modules (method). This paradigm imposes better control and protection over internal data, and helps to structure complex applications designed and implemented by a team of programmers. An example of an object-oriented environment can be found in US 5,603,031. This discloses an environment in which new agents (essentially objects) consisting of data and program modules can be sent between machines.

While the object-oriented paradigm represents a significant advance in software engineering, the data and modules that constitute each object are static. The paradigm is still inadequate for writing programs that must evolve during execution, eg programs that need to pick up, drop, or substitute selected modules. There have been several attempts at overcoming this limitation. For example, work described in US Patents 4954941, 5175828, 5339430 and 5659751 address techniques for re-linking or re-binding selected software modules dynamically during runtime. Also Microsoft's Win32 provides for explicit mapping and un-mapping of dynamic linked libraries into the address space of a process through the LoadLibrary and FreeLibrary calls. With this prior art, however, the prototype or specification of functions and symbols are fixed beforehand and compiled into application programs. This means that an object cannot invoke a module of another object for which the specification is not known at compile time.

Another shortcoming of both traditional procedural and object-oriented paradigms is that programs consist only of data and program modules, but not the transient information that capture the entire state of execution during runtime. This makes it difficult to interrupt a running program and to migrate it to another machine. Generally it is only possible to transfer the saved data file of completed applications. As a very simple example of this problem, while a word processing document file or a spreadsheet file may be transferred from one machine to another, it is not possible to do so while the file is currently being worked on without reverting to a saved version. Transient information is lost. In the case of a word processing file, for example, this means that any "undo editing" function cannot be used by the receiving machine on the file it has just received because the transient "undo" information is not part of the data file.

At present, such migrations have been effected from outside the program, as utilities in the computing environment. Examples include Amoeba, Charlotte, Sprite and Condor. Such utilities work by taking a snapshot of the running program (or core dump), and by resuming execution on the recipient machine from the snapshot. Unfortunately the migration utilities have no knowledge of the usage requirements and semantics of the components of the running program, and so there is no way to adapt it to the new computing environment. Consequently, the migrated program most likely cannot run in a different computing environment from the original one, such as when the machines have different devices like displays, hard disks and sound cards. Even in cases where it does, it would not run with the same level of efficiency, for example because the machines have different amounts of main memory.

This question of incompatibility between different machines is a major problem in computing networks. Elements of a process that are adapted to one machine may simply not apply to another which requires a different configuration and the problem is one of the major difficulties in developing a fully mobile computing environment. With current techniques the hardware components of the environment must be fully compatible for processes to be transferred between them, and even if different hardware components are in theory compatible, different user set-ups and configurations can still cause difficulties.

In this specification the following terms will be used with the following meaning:

"First class entity": an object that can be manipulated directly.

"Process": a combination of data, program module(s) and current execution state.

"Execution state": the values and contents of transient parameters such as the contents of registers, frames, counters, look-up tables and the like.

According to the present invention there is provided a method for migrating a computing process from a first host to a second host, wherein said process discards data, and/or program code and/or execution states specific to the first host, and wherein said process receives data, and/or program code and/or execution states specific to said second host.

By means of this arrangement a process can adapt from the environment of the first host to the environment of the second host by losing system specific information relating to the first host, and by acquiring system specific information relating to the second host.

In a first embodiment of the invention the process discards data and/or program code and/or execution states specific to the first host prior to migration to the second host. In a preferred embodiment, prior to migration a construct is formed comprising application specific data, and/or program code and execution states of the process. This construct may be formed by a construct operation that suspends all active threads of the process and records application specific data, and/or program code and/or execution states of the process. The construct may comprise only data, program code and execution states falling within lists that are passed to the construct operation. The construct may be provided with an authorizing signature.

After it is formed, the construct may either be sent directly to the second host, or it may be sent to an intermediate memory storage means and from there to the second host when required. When the construct has been sent to the second host, a second process is run on the second host that comprises the data, program code and execution states in the construct.

In this embodiment a third process may be created containing system specific data, and/or program code and/or execution states relating to the second host, and the second process may then assimilate this third process.

In another embodiment the process may discard data, and/or program code and/or execution states specific to the first host after migration to the second host, but before receiving data, and/or program code and/or execution states specific to said second host.

In this embodiment prior to migration a construct is formed comprising data, and/or program code and/or execution states of the process. The construct may be formed by a construct operation that suspends all active threads of the process and records data, and/or program code and/or execution states of the process. The construct may be provided with an authorizing signature.

After formation the construct may be sent directly to the second host, or may by sent via an intermediary memory storage means and then from there to the second host when required. After the construct is sent to the second host a second process is created on the second host comprising the data, program code and execution states in the construct. The second process may then perform a mutate operation that suspends all active threads of the second process and discards system specific data, and/or program code and/or execution states relating to the first host. The second process may comprise only data, program code and execution states falling within lists that are passed to the mutate operation. In this embodiment a third process is created in the second host containing system specific data, and/or program code and/or execution states relating to the second host, and the second process may then assimilate the third process.

In a third embodiment of the invention the process may discard data, and/or program code and/or execution states specific to the first host after migration to the second host and after receiving data, and/or program code and/or execution states specific to the second host.

In this embodiment prior to migration a construct is formed comprising data, and/or program code and/or execution states of the process. The construct may be formed by a construct operation that suspends all active threads of the process and records data, and/or program code and/or execution states of the process. The construct may be provided with an authorizing signature.

After formation the construct may be sent directly to the second host, or may by sent via an intermediary memory storage means and then from there to the second host when required. After the construct is sent to the second host a second process is created on the second host comprising the data, program code and execution states in the construct.

A third process may be created on the second host containing system specific data, and/or program code and/or execution state relating to the second host and the second process may then assimilate this third process. Following this assimilation, the second process may perform a mutate operation that suspends all active threads of the second process and discards system specific data, and/or program code and/or execution states relating to the first host. The second process may comprise only data, program code and execution states falling within lists that are passed to the mutate operation.

In all embodiments of the invention the data, and/or program code and/or execution states discarded by the process may relate to library modules and/or input/output device drivers of the first host, and correspondingly the data, and/or program code and/or execution states received by the process may relate to library modules and/or input/output device drivers of the second host.

It will also be understood that the term "host" should be read broadly as meaning any form of computing environment ranging from a single machine of any kind, to any form of network.

Some embodiments of the invention will now be described with reference to the accompanying drawings, in which:-

Fig.1 is a general schematic model of an operating environment of a computing system,

Fig.2 schematically illustrates a process life-cycle and operations that may be performed on a process,

Fig.3 is a flow-chart illustrating the Hibernaculum Construct operation,

Fig.4 is a flow-chart illustrating the Assimilate operation, and

Fig.5 is a flow-chart illustrating the Mutate operation.

Figure 1 shows the general model of a computing system. An application program 30 comprises data 10 and program modules 20. The operating system 60, also known as the virtual machine, executes the application 30 by carrying out the instructions in the program modules 20, which might cause the data 10 to be changed. The execution is effected by controlling the hardware of the underlying machine 70. The status of the execution, together with the data and results that the operating system 60 maintains for the application 30, form its execution state 40.

Such a model is general to any computing system. It should be noted here that the present invention starts from the realisation that all the information pertaining to the application at any time is completely captured by the data 10, program modules 20 and execution state 40, known collectively as the process 50 of the application 30.

The process 50 can have one or more threads of execution at the same time. Each thread executes the code of a single program module at any given time. Associated with the thread is a current context frame, which includes the following components:

- A set of registers
- A program counter, which contains the address of the next instruction to be executed
- Local variables of the module
- Input and output parameters of the module
- Temporary results of the module

In any module A, the thread could encounter an instruction to invoke another module B. In response, the program counter in the current frame is incremented, then a new context frame is created for the thread before it switches to executing module B. Upon completing module B, the new context frame is discarded. Following that, the thread reverts to the previous frame, and resumes execution of the original module A at the instruction indicated by the program counter, i.e., the instruction immediately after the module invocation. Since module B could invoke another module, which in turn could invoke some other module and so on, the number of frames belonging to a thread may grow and reduce with module invocations and completions. However, the current frame of a thread at any given time is always the one that was created last. For this reason, the context frames of a thread are typically stored in a stack with new frames being pushed on and popped from the top. The context frames of a thread form its execution state, and the state of all the threads within the process 50 constitute its execution state 40 in Fig.1.

The data 10 and program modules 20 are shared among all threads. The data area is preferably implemented as a heap, though this is not essential. The locations of the data 10 and program modules 20 are summarized in a symbol table. Each entry in the table gives the name of a datum or a program module, its starting location in the address space, its size, and possibly other descriptors. Instead of having a single symbol table, each process may alternatively maintain two symbol tables, one for data alone and the other for program modules only, or the process could maintain no symbol table at all.

In a preferred embodiment of the present invention, the data and program code of a process are stored in a heap and a program area respectively and are shared by all the threads within the process. In addition the execution state of the process comprises a stack for each thread, each stack holding context frames, in turn each frame containing the registers, local variables and temporary results of a program module, as well as addresses for further module invocations and returns. Before describing an embodiment of the invention in more detail, however, it is first necessary to introduce some definitions of data types and functions that are used in the embodiment and which will be referred to further below.

In addition to conventional data types such as integers and pointer, four new data types Data, Module, Stack and Hibernaculum are defined in the present invention:

Data: A variable of this data type holds a set of data references. Members are added to and removed from the set by means of the following functions;

Int AddDatum(Data d, String dataname) inserts the data item dataname in the heap of the process as a member of d.

Int DelDatum(data d, String dataname) removes the data item dataname from d.

Module: A variable of this data type holds a set of references to program modules. Members are added to and removed from the set with the following functions;

Int AddModule(Module d, String modulename) inserts the program module modulename in the program area of the process as a member of d.

Int DelModule(Module d, String modulename) removes the program module modulename from d.

Stack: A variable of this data type holds a list of ranges of execution frames from the stack of the threads. The list may contain frame ranges from multiple threads, however no thread can have more than one range. Variables of this type are manipulated by the following functions:

Int OpenFrame(Stack d, Thread threadname) inserts into d a new range for the thread threadname, beginning with the thread's current execution frame. This function has no effect if the thread already has a range in d.

Int CloseFrame(Stack d, Thread threadname) ends the open-ended range in d that belongs to the thread threadname. This function has no effect if the thread does not currently have an open-ended range in d.

Int PopRange(Stack d, Thread threadname) removes from d the range belonging to the thread threadname.

Hibernaculum: A variable of this data type is used to hold a suspended process.

As will be explained in more detail below a process may be suspended and stored in a hibernaculum prior to being transferred from one operating environment to another operating environment and/or may be subject to evolutionary operations:

Hibernaculum Construct(Stack s, Module m, Data d): This operation creates a new process with the execution state, program table and data heap specified as input parameters. The process is immediately suspended and then returned in a hibernaculum.

The hibernaculum may be signed by the originating process as indication of its authenticity. Fig.3 is a flow-chart showing the hibernaculum construct operation.

A hibernaculum may be sent between operating environments by the following send and receive functions:

Int Send(Hibernaculum h, Target t) transmits the process contained within h to the specified target.

Hibernaculum Receive(Source s) receives from the specified source a hibernaculum containing a process.

A hibernaculum may be subject to the following evolutionary function:

Int Assimilate(Hibernaculum h, OverrideFlags f) activates the threads of the process stored within h and runs them as threads within a calling process's operating environment. Where there is a conflict between the data and/or program modules of the hibernaculum and the operating environment, the override flags specify which to preserve. Fig.4 is a flow-chart illustrating the steps of the assimilate operation.

Int Mutate(Stack s, int sflag, Module m, int mflag, Data d, int dflag) modifies the execution state, program table and data heap of the calling process. If a thread has an entry in s, only the range of execution frames specified by this entry is preserved, the other frames are discarded. Execution stacks belonging to threads without an entry in s are left untouched. In addition, program modules listed in m and data items listed in d are kept or discarded depending on the flag status. Fig.5 is a flow-chart illustrating the steps of the mutate operation.

Fig.2 illustrates very schematically how these operations may act on a process 230 (which may be loaded from an application 210 or a hibernaculum 220). The process 230 may be subject to a Construct operation 110 to create a hibernaculum, a hibernaculum may be sent to a stream by a Send operation 120, or received from a stream by a Receive operation 130. The contents of a hibernaculum may be assimilated in the process by an Assimilate operation 140, and a process may be caused to mutate by a Mutate operation 150. The process 230 may of course also be subject to traditional operations.

An exemplary embodiment of the invention will now be described in which it is assumed that an executing process p1 is required to migrate from a first host machine t1 to a second host machine t2.

To begin with the process p1 calls the following function:

Hibernaculum h = Construct(Stack s, Module m, Data d)

where s contains all the execution stacks in the process, m contains all the application specific modules in the process (ie m excludes those modules that are system specific to the first host machine), and d contains all the application specific data (ie d excludes data that are system specific to the first host machine). This function creates a new process p2 that contains only the data, program code and execution states of the original process p1 that are specific to the application and not to the system of the first host machine. Process p2 is immediately suspended and returned within a hibernaculum h.

The original process p1 then calls the function:

Int Send(hibernaculum h, Host t2)

which transmits the process p2 contained within hibernaculum h as a stream to the new host machine t2.

At the new host machine t2 a new process p3 is created containing initial system specific data and program code relating to the new host t2. This new process p3 then immediately executes the function:

Hibernaculum h = Receive(Host t1)

which receives from the first host machine t1 the hibernaculum h containing the process p2 which includes only the application specific data, program codes and execution states.

Process p3 then calls the function:

Int Assimilate(Hibernaculum h)

to activate the threads of the process p2 stored in h and to run them as new threads within the environment of the calling process p3. The original thread in p3 then terminates, resulting in a process that has all the application specific portions of process p1, but with the system specific portions replaced to suit the requirements of the second host machine t2.

The original process p1 terminates.

In the embodiment described above the process p1 discards first host specific information prior to migration. However, the discarding may be done after migration to the second host. This is described in the following embodiment.

To begin the process p1 calls the following function:

Hibernaculum h = Construct(Stack s, Module m, Data d)

where s contains all the execution stacks in the process, m contains all modules in the process (including both system specific and application specific modules), and d contains all data in the process (including both system specific and application specific data). Thus the entire process p1 is contained within the hibernaculum as a new process p2 that is identical to p1. Process p2 is immediately suspended and returned within hbernaculum h.

The original process p1 then calls the function:

Int Send(hibernaculum h, Host t2)

which transmits the process p2 contained within the hibernaculum h as a stream to the new host machine t2 where the process p2 is then re-activated. The process p2 then calls the function:

Int Mutate(Stack s, int sflag, Module m, int mflag, Data d, int dflag)

where s contains all the execution stacks in the process, m contains all the application specific modules in the process, and d contains all the application specific data in the process, and where the flags are set to retain the contents of s, m and d. In this way all execution states, and all application specific modules and data are retained in the process p2, but all data and modules specific to the system of the first host are discarded. Process p2 is then stored within a hibernaculum h in the second host using the construct operation.

At the new host machine t2 a new process p3 is created containing initial system specific data and program code relating to the new host t2. This process p3 then calls the function:

Int Assimilate(Hibernaculum h)

to activate the threads of the process p2 stored in h and to run them as new threads within the environment of the calling process p3. The original thread in p3 then terminates, resulting in a process that has all the application specific portions of p1, but with the system specific portions replaced to suit the requirements of the second host machine t2.

It will also be understood that in a variation of this second embodiment the mutate and assimilate functions may be reversed. That is to say, following the transfer of process p2 (identical to p1) from host t1 to host t2, process p2 may assimilate process p3 (containing the t2 system specific data and code) before the mutate operation is used to discard the t1 system specific data and code.

Thus it will be seen that there is provided a method by means of which a process can migrate from one host machine to another host machine and in doing so can adapt to the system requirements and configurations of the second host machine. Such a method allows processes to be readily transferred between host machines thus substantially facilitating the creation of a mobile computing environment.

To implement the process migration system of the present invention in a Java environment, a package called snapshot is introduced. This package contains the following classes, each of which defines a data structure that is used in the migration and adaptation operations:

```
public class Hibemaculum {
          ...
}

public class State {
          ...
}

public class Module {
          ...
}

public class Data {
          ...
}

public class Machine {
          ...
}
```

In addition, the package contains a Snapshot class that defines the migration and adaptation operations:

```
public class Snapshot {
          private static native void registerNatives();
          static {
                registerNatives();
        }

        public static native Hibernaculum Construct(State s, Module m, Data d);
        public static native int Send(Hibernaculum h, OutputStream o);
        public static native Hibernaculum Receive(InputStream i);
        public static native int Assimilate(Hibernaculum h, int f);
        public static native int Mutate(State s, int sflag, Module m, int mflag, Data d, int
        dflag)

        // This class is not to be instantiated
        private Snapshot() {
        }
}
```

The methods in the Snapshot class can be invoked from application code. For example:

```
try {
        if (snapshot.Snapshot.Construct(s, m, d) != null) {
                // hibernaculum has been created
        } else {
                // failed to create hibernaculum
        }
        catch(snapShot.SnapshotException e) {
                // Failed to create hibernaculum
        }
```

The migration and adaptation operations are implemented as native codes that are added to the Java virtual machine itself, using the Java Native Interface (JNI). To do that, a Java-to-native table is first defined:

```
#define KSH “Ljava/snapshot/Hibernaculum;”
#define KSS “Ljava/snapshot/State;”
#define KSM “Ljava/snapshot/Module;”
#define KSD “Ljava/snapshot/Data;”

static JNINativeMethod snapshot_Snapshot_native_methods[] = {
        {
                “Construct”,
                “(“KSSKSMKSD”)”KSH,
                (void*)Impl_Snapshot_Construct
        },
        {
                “Send”,
                “(“KSH”Ljava/io/OutputStream;)I”,
                (void*)Impl_Snapshot_Send
        },
        {
                “Receive”,
                “(Ljava/io/InputStream;)”KSH,
                (void*)Impl_Snapshot_Receive
        },
        {
                “Assimilate”,
                “(“KSH”I)I”,
                (void*)Impl_Snapshot_Assimilate
        },
        {
                “Mutate”
                “(“KSSKSM”I”KSD”I)I”
                (void*)Impl_Snapshot_Mutate
        },
};
```

After that, the native implementations are registered via the following function:

```
JNIEXPORT void JNICALL
Java_snapshot_Snapshot_registerNatives(JNIEnv *env, jclass cls) {
        (*env)->RegisterNatives(     env,
                                     cls,
                                     snapshot_Snapshot_native_methods,
                                     sizeof(snapshot_Snapshot_native_methods) /
                                     sizeof(JNINativeMethod)      );
}
```

Besides the above native codes, several functions are added to the Java virtual machine implementation, each of which realizes one of the migration and adaptation operations:

```
void* Impl_Snapshot_Construct(..) {
        // follow flowchart in Figure 3
        ...
}

void* Impl_Snapshot_Send(..) {
        // send given hibernaculum to specified target
        ...
}

void* Impl_Snapshot_Receive(..) {
        // receive a hibernaculum from a specified source
        ...
}

void* Impl_Snapshot_Assimilate(..) {
        // follow flowchart in Figure 4
        ...
}

void*Impl_Snapshot_Mutate(..){
        //follow flowchart in Figure 5
        ...
}
```

CLAIMS

1. A method for migrating a computing process from a first host to a second host, wherein said process discards data, and/or program code and/or execution states specific to the first host, and wherein said process receives data, and/or program code and/or execution states specific to said second host.

2. A method as claimed in claim 1 wherein said process discards data, and/or program code and/or execution states specific to said first host prior to migration to said second host.

3. A method as claimed in claim 2 wherein prior to migration a construct is formed comprising application specific data, and/or program code and/or execution states of said process.

4. A method as claimed in claim 3 wherein said construct is formed by a construct operation that suspends all active threads of said process and records application specific data, and/or program code and/or execution states of said process.

5. A method as claimed in claim 4 wherein said construct comprises only data, program code and execution states falling within lists that are passed to said construct operation.

6. A method as claimed in any of claims 3 to 5 wherein said construct is provided with an authorizing signature.

7. A method as claimed in any of claims 3 to 6 wherein said construct is sent directly to said second host on a communication medium.

8. A method as claimed in any of claims 3 to 6 wherein said construct is sent to an intermediary memory storage means, and subsequently to said second host.

9. A method as claimed in any of claims 7 to 8 wherein a second process is run on said second host that comprises the data, program code and execution states in said construct.

10. A method as claimed in claim 9 wherein in said second host a third process is created containing system specific data, and/or program code and/or execution states relating to said second host, and wherein said second process assimilates said third process.

11. A method as claimed in claim 1 wherein said process discards data, and/or program code and/or execution states specific to said first host after migration to said second host, but before receiving data, and/or program code and/or execution states specific to said second host.

12. A method as claimed in claim 11 wherein prior to migration a construct is formed comprising data, and/or program code and/or execution states of said process.

13. A method as claimed in claim 12 wherein said construct is formed by a construct operation that suspends all active threads of said process and records data, and/or program code and/or execution states of said process.

14. A method as claimed in any of claims 12 to 13 wherein said construct is provided with an authorizing signature.

15. A method as claimed in any of claims 12 to 14 wherein said construct is sent directly to said second host on a communication medium.

16. A method as claimed in any of claims 12 to 14 wherein said construct is sent to an intermediary memory storage means, and subsequently to said second host.

17. A method as claimed in any of claims 15 to 16 wherein a second process is created on said second host that comprises the data, program code and execution states in said construct.

18. A method as claimed in claim 17 wherein said second process performs a mutate operation that suspends all active threads of said second process and discards system specific data, and/or program code and/or execution states relating to said first host.

19. A method as claimed in claim 18 wherein said second process comprises only data, program code and execution states falling within lists that are passed to said mutate operation.

20. A method as claimed in claim 19 wherein in said second host a third process is created containing system specific data, and/or program code and/or execution states relating to said second host, and wherein said second process assimilates said third process.

21. A method as claimed in claim 1 wherein said process discards data, and/or program code and/or execution states specific to said first host after migration to said second host, and after receiving data, and/or program code and/or execution states specific to said second host.

22. A method as claimed in claim 21 wherein prior to migration a construct is formed comprising data, and/or program code and/or execution states of said process.

23. A method as claimed in claim 22 wherein said construct is formed by a construct operation that suspends all active threads of said process and records data, and/or program code and/or execution states of said process.

24. A method as claimed in any of claims 22 to 23 wherein said construct is provided with an authorizing signature.

25. A method as claimed in any of claims 22 to 24 wherein said construct is sent directly to said second host on a communication medium.

26. A method as claimed in any of claims 22 to 24 wherein said construct is sent to an intermediary memory storage means, and subsequently to said second host.

27. A method as claimed in any of claims 25 to 26 wherein a second process is created on said second host that comprises the data, program code and execution states in said construct.

28. A method as claimed in claim 27 wherein in said second host a third process is created containing system specific data, and/or program code and/or execution states relating to said second host, and wherein said second process assimilates said third process.

29. A method as claimed in claim 28 wherein said second process performs a mutate operation that suspends all active threads of said second process and discards system specific data, and/or program code and/or execution states relating to said first host.

30. A method as claimed in claim 29 wherein said second process comprises only data, program code and execution states falling within lists that are passed to said mutate operation.

31. A method as claimed in any preceding claim wherein the data, and/or program code and/or execution states discarded by said process relate(s) to library modules and/or input/output device drivers of said first host.

32. A method as claimed in any preceding claim wherein the data, and/or program code and/or execution states received by said process relate(s) to library modules and/or input/output device drivers of said second host.

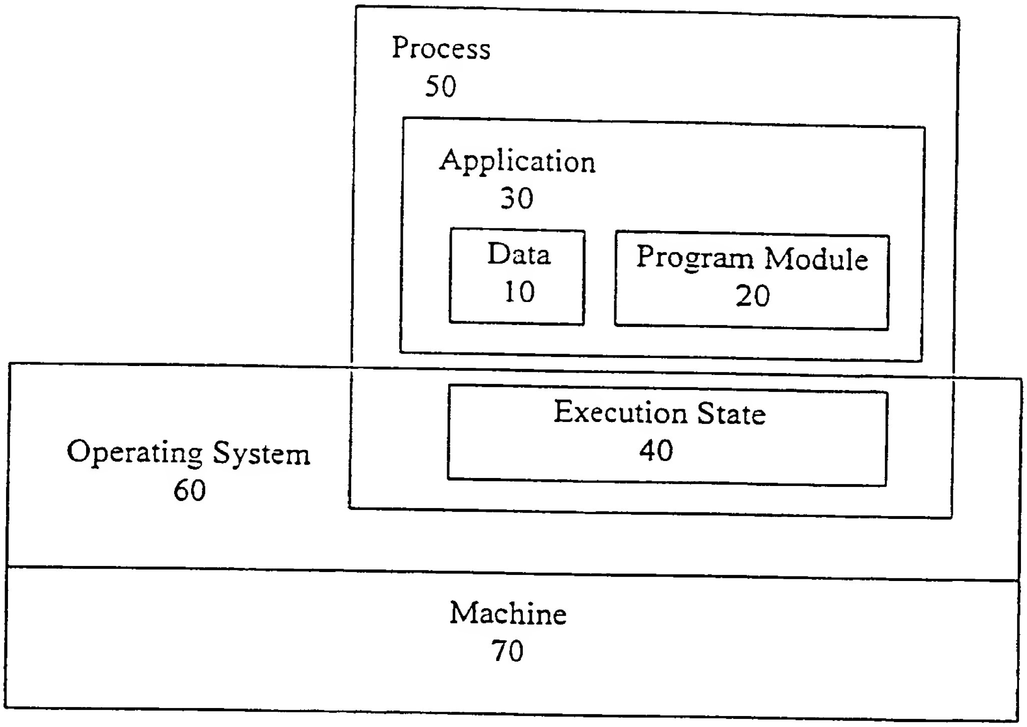

Process
50
Application
30
Data
10
Program Module
20
Operating System
60
Execution State
40
Machine
70

Fig.1

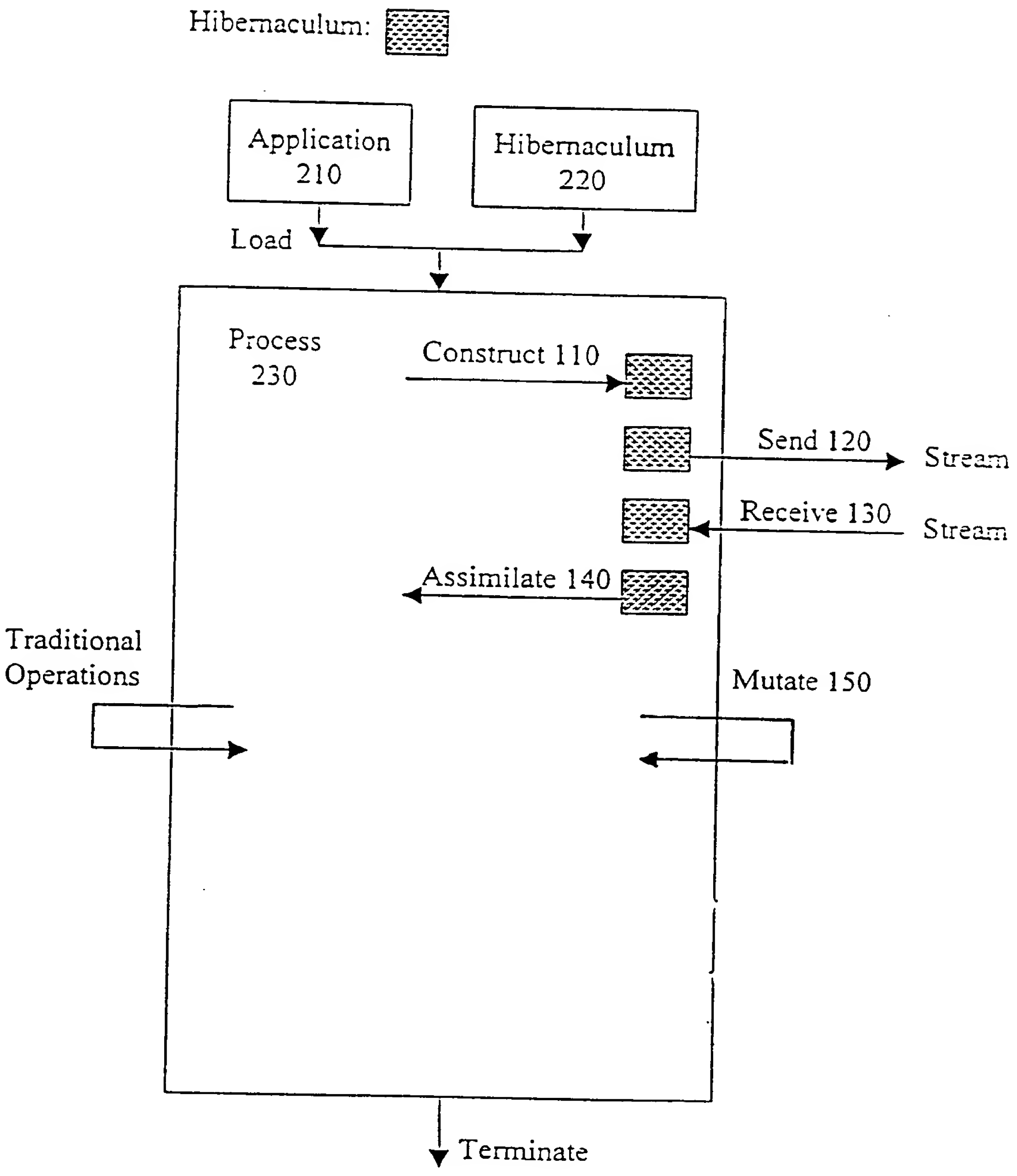

Hibernaculum: ▨
Application 210
Hibernaculum 220
Load
Process 230
Construct 110
Send 120
Stream
Receive 130
Stream
Assimilate 140
Traditional Operations
Mutate 150
Terminate

Fig.2

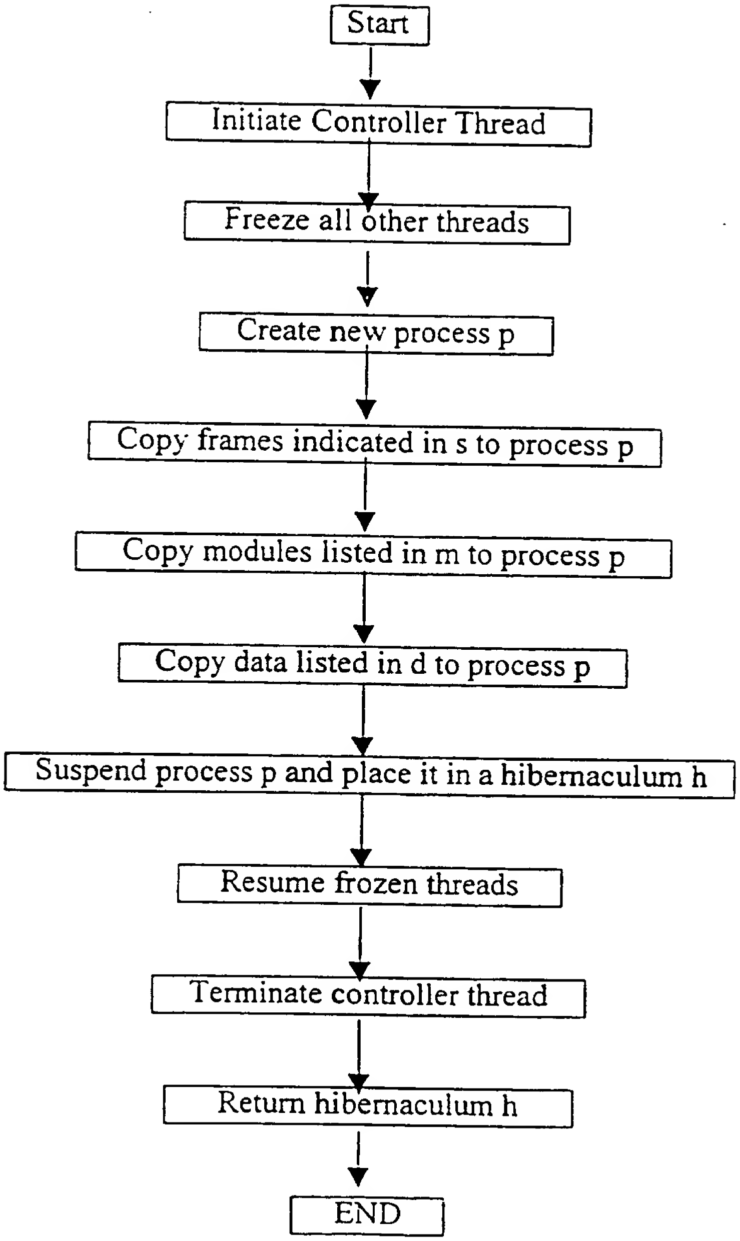

Start
Initiate Controller Thread
Freeze all other threads
Create new process p
Copy frames indicated in s to process p
Copy modules listed in m to process p
Copy data listed in d to process p
Suspend process p and place it in a hibernaculum h
Resume frozen threads
Terminate controller thread
Return hibernaculum h
END

Fig.3

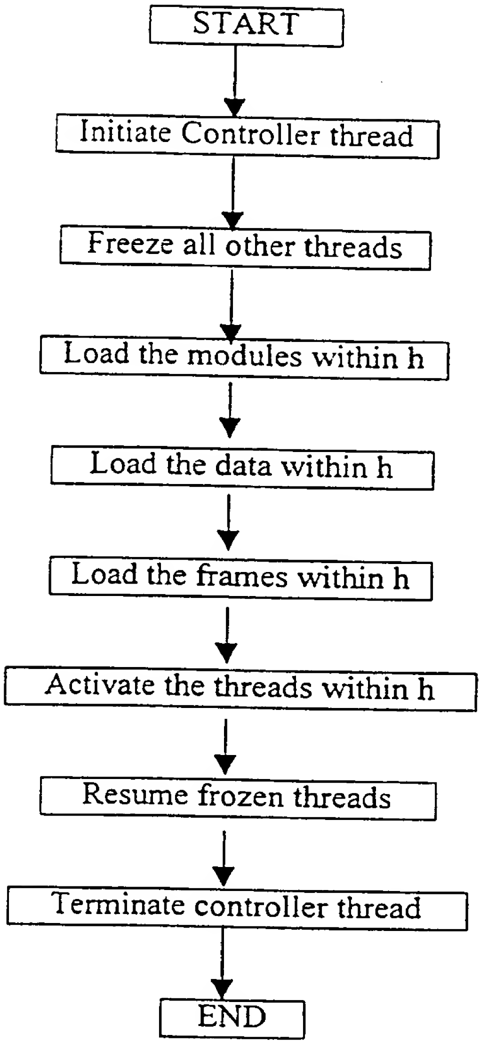

START
Initiate Controller thread
Freeze all other threads
Load the modules within h
Load the data within h
Load the frames within h
Activate the threads within h
Resume frozen threads
Terminate controller thread
END

Fig.4

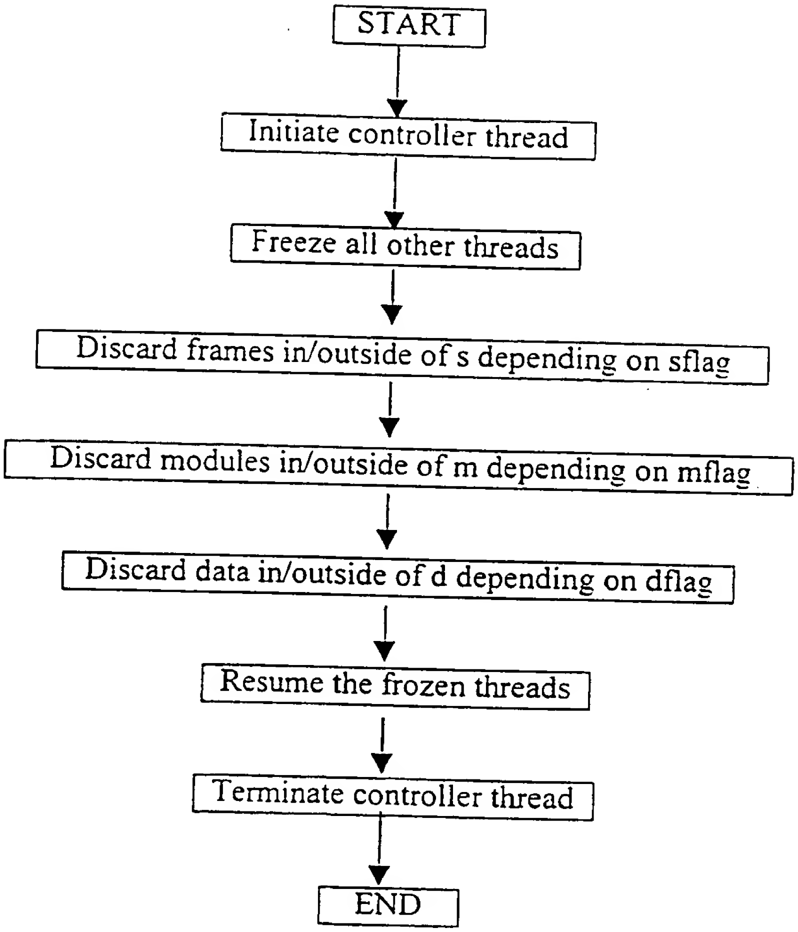

START
Initiate controller thread
Freeze all other threads
Discard frames in/outside of s depending on sflag
Discard modules in/outside of m depending on mflag
Discard data in/outside of d depending on dflag
Resume the frozen threads
Terminate controller thread
END

Fig.5

INTERNATIONAL SEARCH REPORT

International application No.
PCT / SG 99/00018

A. CLASSIFICATION OF SUBJECT MATTER

IPC^6: G06F 9/46, G06F 9/54

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC^6: G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPODOC, WPI, The INTERNET, ACM Digital Library

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 5603031A (HELGESON et al.) 11 February 1997 (11.2.1997), summary of the invention, col. 7 li. 66 - col.8 li. 10, col. 8 li. 51 ff., col. 10 li. 31- col. 11 li. 28 , fig. 24A-C, totality.	1-10, 31,32 11-30
X A	ARIDOR, Yariv, and LANGE Danny B. Agent design patterns: elements of agent application design. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p.108-115 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/articles/proceedings/ai /280765/p108-aridor/p108-aridor.pdf>.	1-3,7,9,31,32 4-6,8,10-30
A	CHEN Qiming, CHUNDI Parvathi, DAYAL Umeshwar, and HSU Meichun. Dynamic software agents for business intelligence applications (poster). Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p.453-454 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:<http://www.acm.org/pubs/ articles/ proceedings/ai/280765/p453-chen/p453-chen.pdf>.	1-32

☒ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:

„A" document defining the general state of the art which is not considered to be of particular relevance

„E" earlier application or patent but published on or after the international filing date

„L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

„O" document referring to an oral disclosure, use, exhibition or other means

„P" document published prior to the international filing date but later than the priority date claimed

„T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

„X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

„Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

„&" document member of the same patent family

Date of the actual completion of the international search 18 May 2000 (18.05.2000)	Date of mailing of the international search report 19 May 2000 (19.05.00)
Name and mailing adress of the ISA/AT Austrian Patent Office Kohlmarkt 8-10; A-1014 Vienna Facsimile No. 1/53424/535	Authorized officer Fastenbauer Telephone No. 1/53424/447

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG99/00018

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CHAUHAN Deepika and BAKER Albert D. JAFMAS: a multiagent application development system. Proceedings of the second international conference of Autonomous agents, Minneapolis, MN USA, p. 100-107 [online], May 10-13,1998 [retrieved on 2000-05-17]. Retrieved from the Internet URL:< http://www.acm.org/pubs/citations/proceedings /ai/280765/p100-chauhan/ p100-chauhan.pdf>.	1-32
A	PKZIP (R) FAST! Create/Update Utility Version 2.04g, 1993-01-02. Copyright 1989-1993 PKWARE Inc. Shareware Version.	1-32
A	WO 97/35262 A (HITACHI), 25 September 1997 (25.9.1997), totality.	1-32
A	BS2000/OSD-BC V1.0, Dynamic Binder Loader/Starter, Chapter 2.3.1: Unloading and unlinking objects (p.44), ID: U5137-J-Z125-2-7600 [online] April 1993 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL:http://manuals.mchp.siemens.de/servers/bs2_man/man_us/bs2_bc/ V1_0/bls.pdf>	1-10,31,32
A	openUTM V4.0 (BS2000/OSD), Concepts and Functions, Chapter 5: Structure of UTM Applications, especially 5.2: The process concept, para. 3, page 67 , ID: U20683-J-Z135-2-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http:// manuals. mchp. siemens.de/servers/bs2_man/man_us/utm/v4_0/ utm_kon.pdf>	1-10,31,32
A	BS2000/OSD-BC V1.0, Performance Handbook, Chapter 5.3.1: Managing the resource main memory, especially: Deactivation (p.248), Waiting Time runout Control (p.255),Paging management algorithms (pp.256-257), ID: U1794-J-Z125-6-7600 [online] February 1997 [retrieved on 1999-12-13]. Retrieved from the Internet via <URL: http://manuals.mchp.siemens.de/servers/bs2_man/man_us /bs2_bc/V1_0/perform.pdf>	1-10,31,32

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG 99/00018

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. □ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. □ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. □ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

cl. 1	...	a method for migrating a computing process ... discards data
cl. 2	...	... discards data ... prior to migration
cl. 3-4	...	... a construct is formed ... / suspend all active threads
cl. 5	...	... falling whithin lists
cl. 6	...	... authorizing signature
cl. 7	...	... sent directly to said second host
cl. 8	...	... sent to an intermediate storage means ...
cl. 9	...	... second process is run on said second host ...
cl. 10	...	... third process ...

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. □ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. □ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. □ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

□ The additional search fees were accompanied by the applicant's protest.

□ No protest accompanied the payment of additional search fees.

continuation of first sheet (1):

cl. 11-17	... discards data ... after migration
cl. 21-30	... discards data ... after migration ... and after receiving data ...
cl. 31	... relates to library modules and/or input/output drivers of said first host
cl. 32	... relates to library modules and/or input/output drivers of said second host

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/SG 99/00018

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US A 5603031	11-02-1997	AU A1 72164/94	06-02-1995
		EP A2 634719	18-01-1995
		EP A3 634719	03-01-1996
		JP A2 7182174	21-07-1995
		JP T2 7509799	26-10-1995
		WO A1 9502219	19-01-1995
		US A 6016393	18-01-2000
WO A 9735262a		none	